BigARTM: Open Source Library for Regularized Topic Modeling of Large Text Collections

Konstantin Vorontsov[1,3], Oleksandr Frei[2], Murat Apishev[3], and Peter Romov[4]

[1] Department of Intelligent Systems at Dorodnicyn Computing Centre of RAS, Moscow Institute of Physics and Technology, `voron@forecsys.ru`
[2] Schlumberger . . .
[3] Moscow State University, . . .
[4] Yandex . . .

Abstract. BigARTM

Keywords: probabilistic topic model, Probabilistic Latent Sematic Analysis, Latent Dirichlet Allocation, stochastic matrix factorization, regularization, EM-algorithm, BigARTM.

1 Introduction

[?] [?] [?]
[?] [?] [?]

2 Multimodal regularized topic model

multimodal LDA (mmLDA) and correspondence LDA (Corr-LDA) [?]
Matching Words and Pictures, MoM-LDA [?]
[?]
[?]
[?]

3 Online topic modeling

[?]
[?]

4 BigARTM architecture

4.1 Algorithm

Algorithm 1 Online EM-algorithm

```
1: Initialize $\phi_{wt}$ for all $w \in W$ and $t \in T$;
2: $n_{wt} := 0$, $\tilde{n}_{wt} := 0$
3: for all batches $D_j$, j = 1,...,J do
4:     $\tilde{n}_{wt} := \tilde{n}_{wt} + ProcessBatch(D_j, \phi_{wt})$
5:     if (synchronize) then
6:         $n_{wt} := Merge(n_{wt}, \tilde{n}_{dw})$;
7:         $r_{wt} := Regularize(n_{wt})$;
8:         $\phi_{wt} := Normalize(n_{wt} + r_{wt})$;
9:         $\tilde{n}_{dw} := 0$;
```

Algorithm 2 $ProcessBatch(D_j, \phi_{wt})$

```
Require: Batch $D_j$, Matrix $\phi_{wt}$;
Ensure: $n_{wt}$;
1: $n_{wt} := 0$ for all $w \in W$ and $t \in T$;
2: for all $d \in D_j$ do
3:     initialize $\theta_{td}$ for all $t \in T$;
4:     repeat
5:         $Z_w := \sum_{t \in T} \phi_{wt}\theta_{td}$ for all $w \in d$;
6:         $\theta_{td} := \frac{1}{n_d} \sum_{w \in d} n_{dw}\phi_{wt}\theta_{td}/Z_w$ for all $t \in T$;
7:     until $\theta_d$ converges;
8:     increment $\tilde{n}_{wt}$ by $n_{dw}\phi_{wt}\theta_{td}/Z_w$ for all $w \in d$ and $t \in T$;
```

4.2 Concurrency

BigARTM can utilize multiple CPU cores by processing several batches in parallel, which follows the rule of expressing parallelism at the highest possible level. A alternative would be to parallelize the $ProcessBatch(D_j, \phi_{wt})$ routine, but keeping it simple has many obvious benefits — single-threaded code is much easier to modify and optimize.

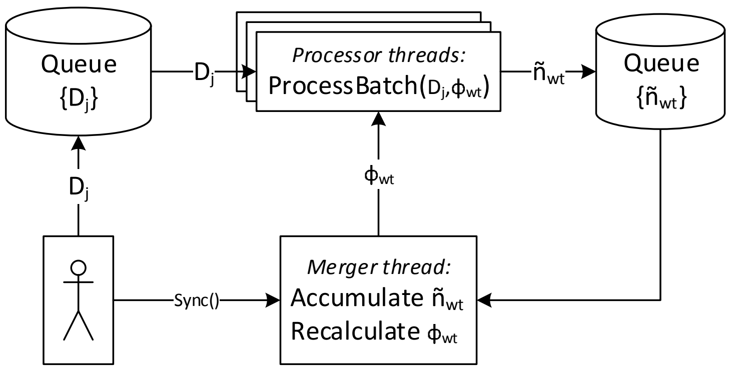

Fig. 1. Diagram of key BigARTM components

Inputs and the outputs of $ProcessBatch$ routine are stored in two in-memory queues, locked for push and pop operations with spin locks. This approach does not add any noticeable synchronization overhead because both queues only store smart pointers to the actual data objects, so push and pop operations does not involve copying of real memory.

Smart pointers are also essential for handling of the ϕ_{wt} matrix. This matrix is *read* by all processors threads, and can be *written* at any time by the merger thread. To resolve this conflict we keep two copies of the ϕ_{wt} — an *active* Φ and a *background* Φ matrices. The active matrix is read-only, and is used by the processor threads. The background matrix is being built in a background by the merger thread from n_{wt} and r_{wt} counters, and once it is ready merger thread marks it as active. Before processing a new batch the processor threads gets the current active matrix from the merger thread. This object is passed via shared smart pointer to ensure that processor thread can keep ownership of the ϕ_{wt} matrix until the batch is fully processed.

Having two copies of the Φ matrix is an obvious drawback of this solution. An alternative would be to use atomic CPU instructions to read and write values of the Φ matrix. Such operations are very efficient, but still come at a considerable synchronization cost [5]. Using them for all reads and writes of the Φ matrix would cause significant performance degradation for merger and processor threads. Besides, an arbitrary overlap between reads and writes of the Φ matrix eliminates any possibility of producing a deterministic result. The design with two copies of the Φ matrix give much more control over this and in certain cases allows BigARTM to behave in a fully deterministic way.

The design with two Φ matrices only support a single merger thread, and we believe it should cape with all $\hat{n}_{wt}$ updates coming from many threads. This is a reasonable assumption because $Merge$ routine takes only about $O(W * T)$ operations to execute, while $ProcessBatch()$ takes $O(N_{nz} * T * I)$ operations, where W is the size of the dictionary, N_{nz} is the number of non-zero entries in the batch, T is the number of topics, and I is the averate number of inner iterations in $ProcessBatch()$ routine. N_{nz}/W is typically 100 to 1000 (based on datasets in UCI Bag-Of-Words repository), and I is $10 \dots 20$, so the ratio safely exceeds the expected number of cores (up to 32 physical CPU cores in modern workstations, and even 60 cores if we are talking about Intel Xeon Phi co-processors).

4.3 Memory layout

4.4 Programming interface

The core of BigARTM is written in C++, but we decided to wrap it into an "extern C" API. This makes it very easy to consume BigARTM from almost any programming environment, because there is always a way of calling "extern C" code (ctypes module for Python, loadlibrary for Matlab, PInvoke for C#, etc).

[5] http://stackoverflow.com/questions/2538070/atomic-operation-cost

5 Experiments

6 Conclusions

Acknowledgements. The work was supported by the Russian Foundation for Basic Research grants 14-07-00847, 14-07-00908, and by Skolkovo Institute of Science and Technology (project 081-R).